NASA JPL Distributed Systems Technology (DST) Object-Oriented Component Approach for Software Inter-operability and Reuse

Laverne Hall, Chaw-Kwei Hung
Phone: 818 393–5430
Fax: 818 393-4049
Laverne.Hall@jpl.nasa.gov

Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Dr.
Pasadena, CA 91109

Abstract

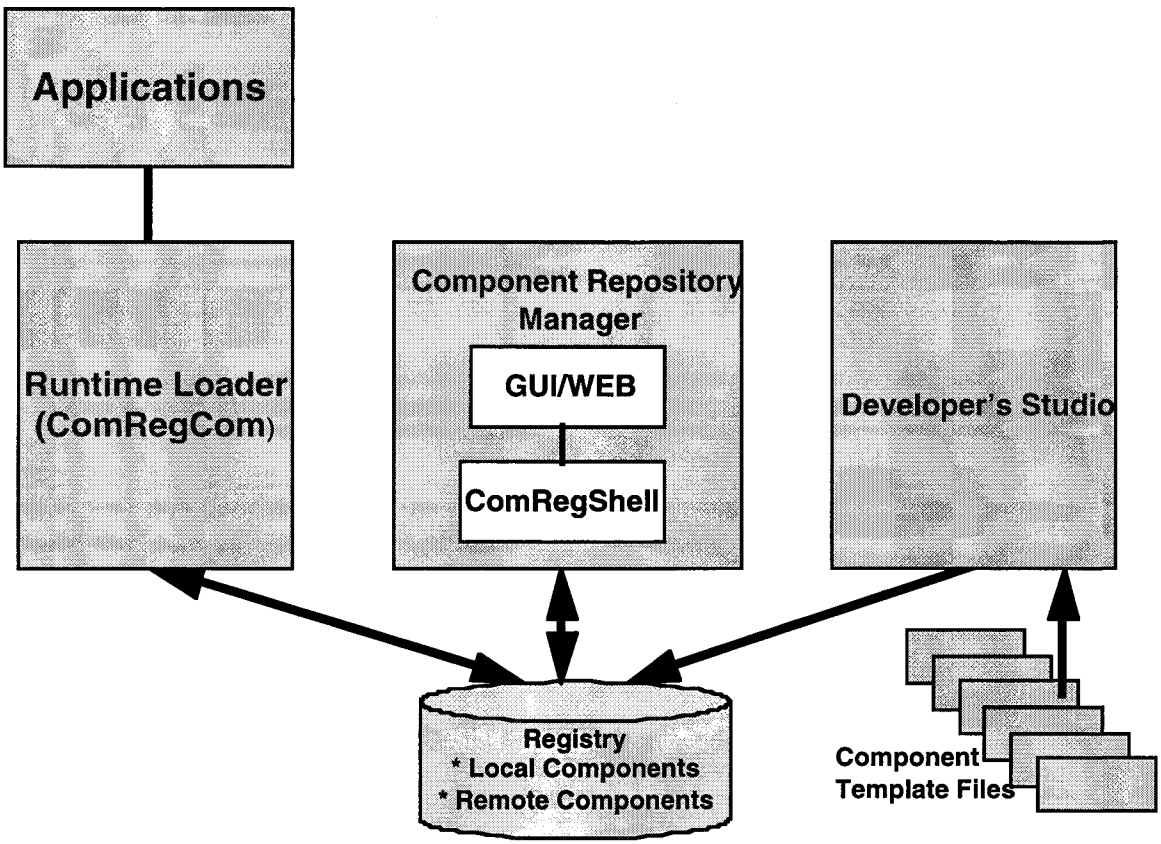
The purpose of this paper is to provide a description of NASA JPL Distributed Systems Technology (DST) Section's object-oriented component approach to open inter-operable systems software development and software reuse. It will address what is meant by the terminology object component software, an overview of the component approach and how it relates to software architecture and reuse, the benefits of this approach, and examples of application prototypes demonstrating its advantages.

The study scope of this paper, which includes rapid prototyping, focuses on current and future of NASA JPL software development as it relates to:
• How object component technology will be used in JPL software development and operational environments
• How it provides a framework for inter-operability and reuse of components across subsystems for system upgrade and future migrations (i.e., establish guidelines for using object/components technology)
• How it reduces development, testing, and maintenance relative to life-cycle cost and time
• How it avoids duplication or re-development efforts through building reusable components and making existing software reusable where appropriate.

The infrastructure to support object component technology will be addressed from various points of view: a) components development, b) application development, c) configuration management (CM), and d) operations.

The required support environment and the procedures for developing new software components and component deployment will also be addressed for a UNIX environment. The Object Component Deployment Management System (CDMS) for the object components for software reuse will be illustrated. It will address the issues of developing components, accessing the components for development and for operations, the component security, and the migration path from the transitional to component-capable configuration management

(CM) system. The CDMS consists of three major elements: (1) Component Repository Manager (2) Runtime Loader and (3) Developer Studio. (See Figure 1 – A Pictorial Architecture of the Entire System.)



Component Repository Manager maintains a collection of components that are available to be linked to applications, and provides methods to access those stored components. Runtime Loader (ComRegCom) provides component interface for applications to dynamically load other components from the repository during runtime. Developer Studio is a development environment that helps software developers to start building a component from templates.

Also a security component has been developed and will be used to verify the authenticity or identity of components. This will allow applications to load components at runtime from remote sources and retain confidence that the received component is safe and from a known user.

Utilization of the object-oriented component technology approach for system development and software reuse will apply to several areas within JPL, and possibly across other NASA Centers, for example:

- NASA/JPL Telecommunications & Mission Operations Directorate (TMOD) Deep Space Network (DSN)
- NASA/JPL Flight Software - Mission Data Systems (MDS)

• Other technology and applications programs [ex., NASA/JPL Technology & Applications Programs (TAP) Distributed Modeling Infrastructure (DMI), Department of Defense (DOD), etc.].

NASA JPL Distributed Systems Technology (DST) Object-Oriented Component Approach for Software Inter-operability and Reuse

Laverne Hall
Distributed Computing & Systems Engineering Group, JPL/Caltech
Pasadena, CA 91109-8099, USA

And

Chaw-Kwei Hung (& Imin Lin, JPL Contractor)
DC&SE Group, JPL/Caltech, 4800 Oak Grove Drive,
Pasadena, CA 91109-8099, USA

ABSTRACT

The purpose of this paper is to provide a description of NASA JPL Distributed Systems Technology (DST) Section's object-oriented component approach to open inter-operable systems software development and software reuse. It will address what is meant by the terminology object component software, give an overview of the component-based development approach and how it relates to infrastructure support of software architectures and promotes reuse, enumerate on the benefits of this approach, and give examples of application prototypes demonstrating its usage and advantages.

Utilization of the object-oriented component technology approach for system development and software reuse will apply to several areas within JPL, and possibly across other NASA Centers.

Keywords: Software reuse, inter-operability, object-oriented, object component software, component-based development, infrastructure, distributed systems, and prototypes.

1. BACKGROUND & INTRODUCTION

The Distributed Computing & Systems Engineering Group within the Network & Distributed Systems Technology (DST) Section of JPL provides advanced research solutions to software architectures via infrastructure development and prototypes using new technology for task insertion and provides development and integration & test support to implementation teams using these solutions. Many solutions are centered around the use of an advanced object-oriented distributed systems approach, currently object-oriented component-based software in particular, with code, templates, supporting framework, and sample application prototypes provided.

The study scope of this paper, which includes rapid prototyping, focuses on current and future NASA JPL software development as it relates to:

- How object component technology can be used in JPL software development and operational environments,
- How it provides a framework for inter-operability and reuse of components across subsystems for system upgrade and future migrations (i.e., establish guidelines for using object/components technology),
- How it reduces development, testing, and maintenance relative to life-cycle cost and time, and
- How it avoids duplication or re-development efforts through building reusable components and making existing software reusable where appropriate.

The framework or infrastructure to support object component technology focuses on a UNIX environment and will be addressed relative to various perspectives through-out several sections of this paper: a) components development, b) application development, c) configuration management (CM), and d) operations.

The required support environment and the procedures for developing new software components and component deployment will also be addressed for a UNIX environment via the Object Component Deployment Management System (CDMS). It will address the issues of developing components, accessing the components for development and for operations, the component security, and the migration path from the transitional to component-capable configuration management (CM) system.

2. COMPONENT SOFTWARE APPROACH

Distributed computing allows modern software structure to occur across distributed networks in an increasingly flexible and effective manner. Software component

technology allows distributed application pieces to flexibly be reused, inter-operate, and evolve over time. Figure 1 shows the evolution (most current progressing towards the left direction) of network infrastructure technology support for distributed computing and how the trend has gone from proprietary methods to continuous improvements using open standards.

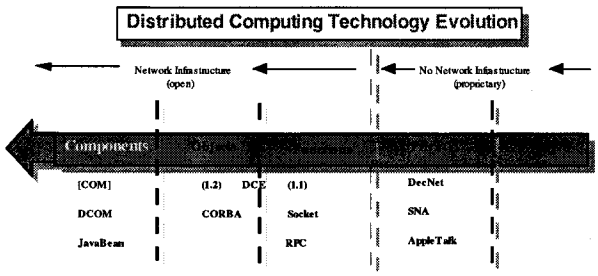


Figure 1. Distributed Computing technology Evolution

The component software approach promotes object-oriented component-based technology and architecture development into software development. It provides a standard framework or infrastructure for building and using software components to save time and money. The tasks promoting this approach also provide prototyped reusable generic object-oriented software components for common services (both communications and application services) needed by many Telecommunications & Mission Operations Directorate (TMOD) Deep Space Network (DSN) subsystems and other Technology & Applications Programs (TAP).

The top portion of Figure 2 gives a pictorial view of the goal to move from a traditional monolithic type application composed of a single binary file to applications which can be easily configured (static or dynamic, local or remote) from a library of components providing common services across applications. Applications would reduce their focus to developing custom or application-specific components to plug-and-play with the generic functional services components.

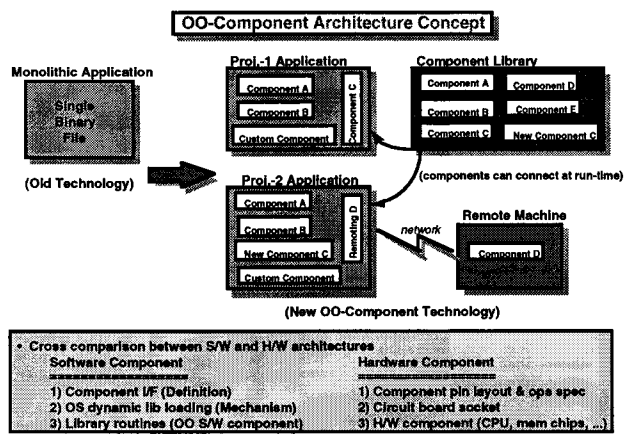


Figure 2. Object Component Architecture Concept

The objective is to design application software from a set of functional software components. Each software component has a well-defined interface that encapsulates a distinct operation. This is a similar process as to constructing a hardware circuit board with the hardware architecture equating to the elements of a software architecture using components (see bottom portion of Figure 2).

A component can be designed to contain a multitude of related functions part of its capabilities (example, communications services component). Figure 3 depicts the fundamental structure of a generic functional component which can provide a variety of related services within the same component allowing different users to configure or initialize the same component to activate preferential modes of sub-services. The component entry point is a register addressing scheme or structure defined by Microsoft's Component Object Model (COM) and in this work, used in a C++ UNIX development environment only. The component entry point indexes to special routines making-up the varied functionality of the component and allows routines to be upgraded or swapped-out of the component without the application setup changing as long as the routine address does not change.

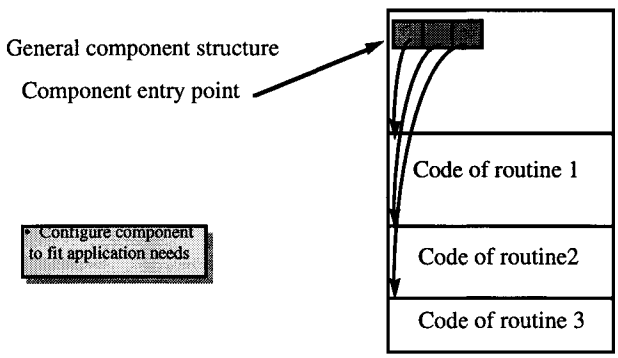


Figure 3. Generic Component Software Structure

Component Software Approach Benefits

All benefits of using the oo-component software approach lead to cost and time savings through-out various phases of the life-cycle of any system:

1) Inter-operability - Execute old with new and across different communication domains; due to need to operate or transition legacy subsystems and custom protocols.
2) Extensibility - Easy to design into new components; new components can reuse existing components.
3) Easy Reuse - Develop once and use many times. Component framework makes reuse easy.
4) Easy Assembly - Components can be added after delivery, like the plug-in capability of Netscape.

5) Runtime Flexibility - Components can be static/dynamic linked and dynamically loaded/swapped as the program runs (Adaptability), provides flexibility of building different software architectures.
6) Enforce Object-Oriented Design (as standard) - Component is really a specialized Object structure (although it can be implemented as non-object).
7) Development Flexibility - allows independent development environments and different implementations of services, if needed.

This approach can allow application configurations to be produced quickly and can result in higher quality, more reliable software.

3. FRAMEWORK & INFRASTRUCTURE

The Distributed Systems Technology (DST) component framework provides C++ UNIX environment development support and allows for the incorporation of different COTS products and middlewares (ex., CORBA or DCE) into the system architecture. The framework includes templates for building the three types of component design patterns which are depicted in Figure 4. The component templates provide developers an easy way to start building a component for their applications. The basic component is the simplest form while the containment pattern allows you to add new functional interface(s) to the component without directly exposing the interface(s) to existing functionality. The aggregation pattern is probably the most efficient and desired method and which minimizes the need for code rebuilds. This pattern exposes existing component interfaces to applications when new functional interfaces are being added. The usage of these template types is fully covered in the User's Guide for this effort.

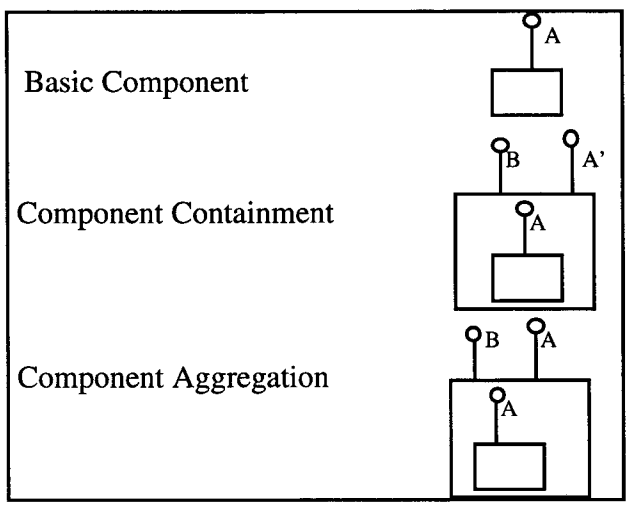


Figure 4. Software Component Design Patterns

All the pseudo names in the template files can be manually replaced. However due to the number of names to be replaced and the number of files involved, our experience tells us that the replacement can become time consuming and error prone. To make the task easier and to eliminate the human operation errors, tools may be developed to automate the entire process (can be part of Developer's Studio in later section on Component Deployment Management System).

The DST component framework can be viewed as layered expandable building blocks of features and services which interact or connect with various other parts of the framework to provide a suite of component development and runtime support capabilities for a variety of software applications and architectures (see Figure 5). Many support features listed in the diagram (such as component locating, retrieving, loading, registration, basic security or authentication, etc.) have been prototyped and demonstrated by DST team-members. Other support features listed in the figure are under research and design considerations as a part of ongoing and future developments.

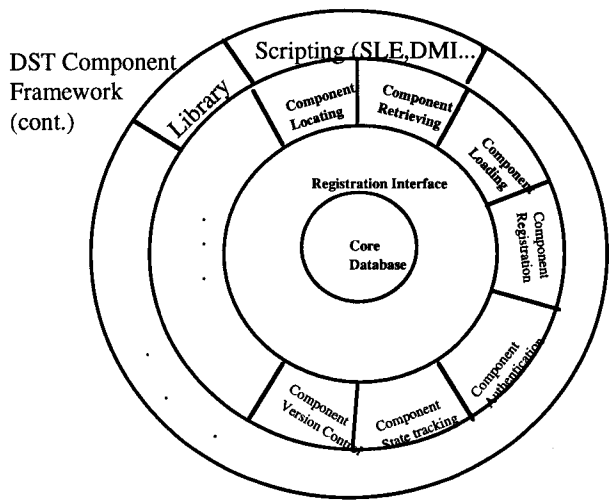


Figure 5. DST Component Framework

The investigation of commercial-off-the-shelf (COTS) software continues for products to support the component framework in a UNIX environment (SoftwareAG, COM/DCOM, etc.). Other COTS being looked at for possible utilization include Rational Rose, via the Unified Modeling Language (UML), to create in-process component templates and to generate modeling scripts (example, to be used by our Distributed Modeling Infrastructure - DMI task which connects separate simulation tools into a cohesive framework for end-to-end space mission simulation).

4. APPLICATIONS & PROTOTYPES

Utilization of the object-oriented component technology approach for system development and software reuse will apply to several areas within JPL, and possibly across other NASA Centers, for example:

- NASA/JPL Telecommunications & Mission Operations Directorate (TMOD) Deep Space Network (DSN)
- NASA/JPL Flight Software - Mission Data Systems (MDS)
- Other technology and applications programs [ex., NASA/JPL Technology & Applications Programs (TAP) Distributed Modeling Infrastructure (DMI), Department of Defense (DOD), etc.].

The DST systems engineering and prototyping team has developed and demonstrated actual component object code for several generic services which can be configured for use by many applications within scientific space and ground systems. Software components for generic communications input/out, smart monitor & control, data manipulation via a symbol table and expression evaluator, security, constraints, and so on have been developed by reusing existing code, incorporating both COTS and in-house implementations for rapid prototype cost and time savings.

Generic Monitor & Control (M&C) - Example

The example shown in this subsection consists of reusable components configured for smart monitoring and control in any application requiring such functionality. The two components, Publish/Subscribe and Symbol Table/Expression Evaluator, used in this example and their interfaces are depicted in Figure 6 using conventional object-interface diagrams.

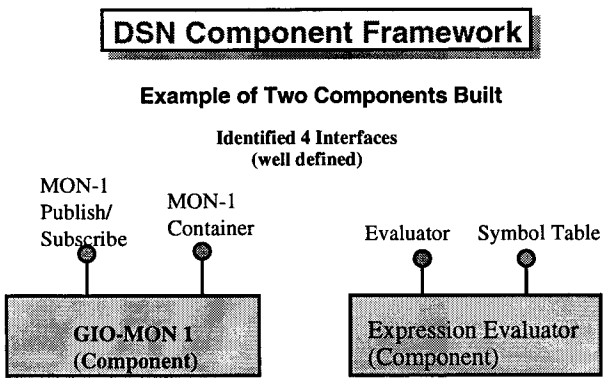


Figure 6. Example of Two Components Built

The functionality or services provided by the two components can be described as follows:

- GIOMON1 (Generic I/O for M&C within the DSN)
  - Publish and Subscribe: Provides the DSN Common Software Monitor and Control Data Publishing and Data Subscribing functions. GIOMON1 is implemented by using the DSN Monitor and Control Infrastructure Services (MCIS) Common Software.

- EVALUATOR (supports smart M&C)
  - Symbol Table and Expression Evaluator: Provides an object to hold a collection of data of various types. All data in Symbol Table can be used as operand in expressions that can be evaluated by Evaluator.

A simple typical smart M&C operational scenario for using these components across remote, local, or same machines (say for spacecraft subsystems M&C) is illustrated in Figure 7 and summarized as follows:

- The Monitor Data Server is a focal-point for the collection and distribution of various types of information on subsystems within the DSN.
- One DSN subsystem (left-most) periodically publishes a list of monitor data items (ex., spacecraft power, temperature, and memory utilization) to the MDS.
- Another subsystem (bottom-most) publishes a policy, at given times, of how the data is to be used; i.e., what decision or action should take place under certain conditions.
- Any subsystem subscribing to continuously updated data and/or policies of interest published to MDS use the data and policy (stored in a symbol table) to perform an expression evaluation (data plugged into policy) to determine appropriate commands based on results (ex., spacecraft power below certain threshold, switch to backup supply).

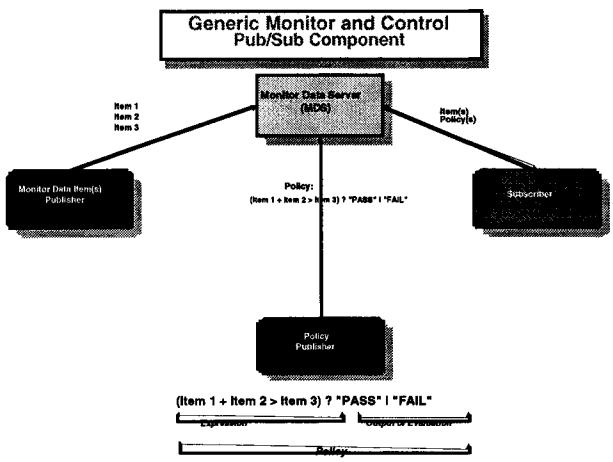


Figure 7. Sample Application - Generic M&C

5. COMPONENT DEPLOYMENT MANAGEMENT SYSTEM

The required support environment and the procedures for developing new software components and component

deployment will also be addressed for a UNIX environment. The Object Component Deployment Management System (CDMS) for the object components for software reuse is a system that provides a central repository for all the components and the interface to access the repository. It will address the issues of developing components, accessing the components for development and for operations, the component security, and the migration path from the transitional to component-capable configuration management (CM) system. The CDMS consists of three major elements: (1) Component Repository Manager (2) Runtime Loader and (3) Developer Studio (see Figure 8 for a depiction of the architecture for the entire system).

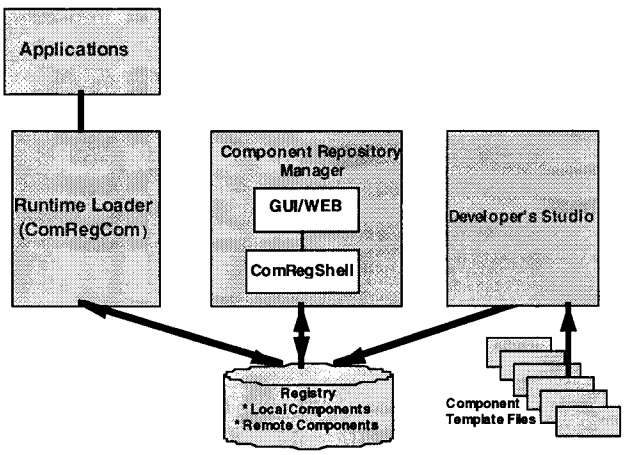


Figure 8. Component Deployment Management System

These three subsystems may work stand alone to suit specific need or be integrated as a suite to provide complete functionality.

As the core part of the Runtime Loader, ComRegCom is a special component that handles the registration and dynamic loading of the available components in a system, which may consist of multiple hosts in a network. Runtime Loader (ComRegCom) provides component interface for applications to dynamically load other components from the repository during runtime.

The component registry is a file (database for future work) where users can specify the name, the location, and other information for a component. The registry API allows ComRegCom to access the component registry so it can find the location of a component and load the component in during runtime.

All the components that are to be used must be specified in the component registry. Then the applications can instanciate the components via CoCreateInstance() function call provided by ComRegCom. This function call dynamically loads the desired component during runtime.

Component Repository Manager maintains a collection of components that are available to be linked to applications, and provides methods to access those stored components.

ComRegShell is a lower layer in Component Repository Manager. It provides API to access the component repository that contains the registry and available components. A upper layer front end, which can be a friendly graphical user interface or an application, can be built on top of ComRegShell to allow user to manage the repository.

The Developer Studio is a development environment that helps software developers to start building a component from templates. The Developer's Studio is a future area of development which can use the three component template types mentioned in the previous Framework & Infrastructure section of this paper.

Also a Security Component has been developed and will be used to verify the authenticity or identity of components. This will allow applications to load components at runtime from remote sources and still retain confidence that the received component is safe, i.e. from a known user and has not been altered in any way.

The detailed design document for CDMS will include details on the overall system architecture (software hierarchy or implementation layout), functional description on each software element and how each element interacts with one another. It should also include the design specification on each software element.

6. STATUS AND FUTURE WORK

Utilization of DST's recommendation of the object-oriented component technology approach for system development and software reuse is being incorporated into system design considerations in several areas within JPL. In addition to TMOD/DSN applications, TAP's Distributed Modeling Infrastructure (DMI) task plans to use this approach and incorporate the Component Registration Component (ComRegCom) as one of its core enablers under infrastructure support to collaboratively bring together different mission simulation packages. Also for consideration by other projects, such as Mission Data Systems (MDS), ground simulations can be built as a scripting of components and these components can be used in flight if the platform is compatible or at least it can be mapped into an on-board component.

Quite a bit of work has been completed via rapid prototyping and demonstration of generic reusable services and the supporting reusable infrastructure or framework to promote an object-oriented component approach to developing science-based applications in a UNIX environment. Some planned areas of continuation for the Component Deployment Management System (CDMS) task for framework support include the following:

1. Component Repository Manager – Migrate component repository into database structure.
2. Component Repository Manager - Enhance user interface via GUI or Web.
3. Developer Studio – Add preliminary GUI front-end to drive scripts for component templates.
4. Security Protocol – Wrap higher level security protocols, such as Secure Socket Layer (SSL), into components.

7. ACKNOWLEDGMENTS

We acknowledge the significant technical contributions made to these efforts by Imin Lin, past JPL lead software systems engineer and current part-time support contractor to JPL. Also, thanks are extended to prototyping contributions made by the technical development team-members: Judy Yin, Amalaye Oyake, and Chialan Hwang.

8. CONCLUSIONS

The objective of DST's reuse and object-oriented component approach is to provide advanced technical solutions via software prototypes and supporting framework, allowing tasks to concentrate on their application-specific domain solution(s). Most scientific software system applications can be built with object-oriented components, supporting framework, and design patterns. The use of the component object approach allows for emerging technologies through careful design of interfaces. Major task functional blocks of systems can be implemented as plug-in modules (Components).

9. REFERENCES

The following are references use to support this work:

[1] Dale Rogerson, Inside COM (Microsoft's Component Object Model), Washington: Microsoft Press, 1997.
[2] Dr. Richard Grimes, Professional DCOM Programming, Canada: Wrox Press Ltd., 1997.
[3] Stanley B. Lippman & Josée Lajoie, C++ Primer - 3rd Ed., Addison-Wesley, 1998.
[4] Terry Quatrani, Visual Modeling with Rational Rose & UML, Addison-Wesley, 1998.